Maria version 2.01 Language Description

Updated: 22 May 2007

This is the official description of Maria Version 2.01. In this document we describe the syntax of the scripting language and the structure of the programs. Anyone interested in writing a Maria script should be familiar with the contents of this document. This is NOT intended to teach programming methods, nor do you need to know this in order to run a Maria script. **Note:** This document is copyright 2007 You are free to distribute this document but not to make changes or additions to it. You may include it with other documents and programs that you or others write as long as it remains in a separate file.

The Design and Limitations of the Maria Scripting Language

First of all, why another language (scripting or otherwise) in a world full of languages? Maria is a small, task-specific language intended to impliment behaviors in simple (read primitive), interactive androids. The language is designed to accommodate advances in technology by using a simplified object-oriented approach that hides low level control inside classes that correspond to parts of the human body. If the android cannot, for whatever reason, accomplish a scripted action, it should fail gracefully; not begin behaving in an erratic manner. The script itself should use the built-in testing facilities to query the android as to its capablities; it is, however, up to the script writer to respect the capabilities of the android and work within the confines of those capabilities.

Rather than being strict, I've opted for a fairly flexible interpretation of "object oriented" and have worked towards making the language compact, understandable and easy-to-use rather than a blind adherence to a particular language model. Since Maria scripts are relatively short compared with other languages (Perl, Basic) many of the keywords and operators are fairly verbose. I chose verbose over compact to enhance readability, especially since non-programmers may have an interest in reading the scripts. You'll find that I've freely borrowed elements from Java, C, Smalltalk, Basic, Pascal and other languages. I've also tried to include enough methods (functions) to allow the creation of complex and unique behaviors.

In selecting the level of detail and language terms I've had to make some arbitrary decisions about what was or was not appropriate. I followed a logical, research driven approach but in the end it came down to weighing alternatives. You might disagree with some of the decisions that have been made concerning the implimentation of this language and you may be entirely correct. The utility of any language is dependent upon its standard use. Please resist the urge to muck with Maria as this will only break the language and make it difficult for everyone. I've also included some Classes/Instances for specifically for future expansion (not yet implimented) should the need or desire arise (i.e. RightLeg, LeftLeg, Pubis, Abdomen).

Finally, the Maria scripting language is only intended for creating behaviors for androids; it is not a general purpose programming language. Please keep this in mind should you have any ideas of trying to extend the language. Here are some aspects of the language and its development/use.

1. Object Oriented

This is a simple object-oriented interpreted scripting language. To simplify things, you may not create or instance new classes; only the classes and instances in the language are available for your use. The reason for this decision is to keep the language simple. Classes in Maria are modelled on the real world; World, Body etc. The classes hide the actual nuts-and-bolts of creating the movement or effect. All the script writer must do is to specify what is required and the interpreter/hardware does the rest. The creator/manufacturer of the android is responsible for the low-level programming of the program/hardware interface in some other language (Java, C++) to allow the interpreter to correctly control the android; this aspect is hidden from the Maria script writer inside the object.

2. Event Driven

Events, defined as changes in the state of sensors making up the nervous system of the android, determine the response of the program. These events are acted upon in the event loop (`main`); events are triggered by changes in the nervous system of the android (in most cases instance and class variables). The sensors send back reponses to the script triggering actions. The hardware can be querried by the Maria script to find out the number of discrete levels of each

of the sensors using the **`sensitivity(`**`<sensation instance variable or self>`**`)`** method. The number of sensory levels can be as many as the hardware will allow. The sampling frequency can also be determined with the **`samplingFrequency(`**`<sensation instance variable or self>`**`)`** method.

3. The Nature of Human Movement

Electromyographic recording of the membranous urethral sphincter during ejaculation. Kollberg, S., Petersen, I., and Stener, I. 1962

One thing you'll notice right away is that many of the bodily functions in Maria are encoded as waves with a period, amplitude and duration. This mimics the way the human body works. Heartbeats, brain activity, ejaculation (see above) and peristalsis are examples of biological functions that can be described as waves. Waves that represent electrical activity (EEG, ECG see below) of the nerves tend to be sharp, spike-like waveforms. Muscle movements resulting from these nervous impulses tend to show a more rounded form due to the damping action of the muscle and inertia. With Maria, the shape of the actual waveform (e.g. triangular, square, sinusoid) is determined by the low-level programming done by the creator/manufacturer of the android since this is closely tied to the mechanics of the android and the source of power (electrical, hydralic, pneumatic).

electrocardiogram (ECG)

Mechanical change in humans may be directly measured or inferred by measuring the state of different biological systems such as:

Nerve: Electromyography
Muscle: Ultrasound, MRI, direct observation, inflated balloons
Blood: Photoplethysmography (AC or DC components), heated electrode, sphygmomanometer

While measurements all of these systems produce waveforms, remember that the waveforms in Maria represent only the actual mechanical output of the muscle or in some cases, the effects of increased blood flow (as in vasocongestion or erection). If you base your programming on other waveforms (i.e. nerve impulses) you'll have to translate these into muscle actions. The following will provide you with some basic information concerning muscle actions in humans.

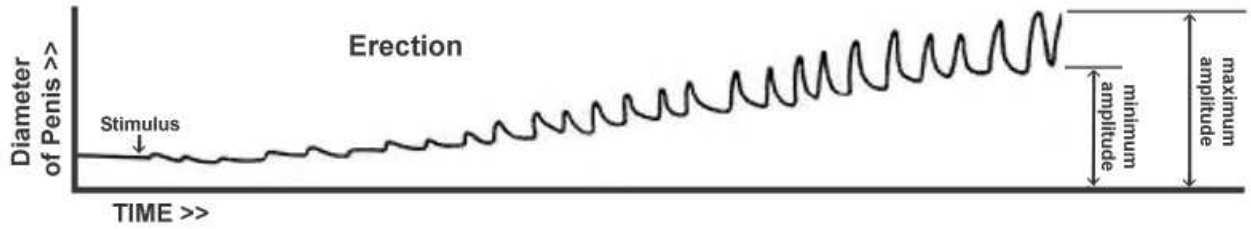
Muscles produce two types of force: isotonic and isometric. An isotonic contraction shortens the muscle but keeps a constant tension. An isometric contraction does not shorten the muscle and allows tension to increase. Most muscle contractions are mixtures of isotonic and isometric contractions.

Muscle contractions may be grouped according to the number and frequency of the nerve impulses. A single nerve impulse gives only 20% to 30% of the maximum tension possible and is called twitch. Several nerve impulses spaced closely enough that the muscle does not have time to relax between the impulses bulds up more tension and is called summation. Summation gives 35% to 100% of the maximum tension possible. If many nerve impulses are given close together the muscle will produce a smooth continuous contraction called tetanus.

Each muscle is made up of thousands of individual motor units. Muscles may increase the amount of tension by using more of the motor units as well as increasing the frequency of the nerve impulses.

Slow-twitch fibers are small and produce twitches up to 100 milliseconds long. Fast-twitch fibers are larger and produce twitches as short as 7.5 milliseconds. After a twitch there is a short refractory period of 5 msec when the muscle cannot respond to another stimulus. If another stimulation arrives before the end of the refractory period, the muscle will achieve summation and achieve a higher level of tension.

Methods controlling the wave use six parameters; period, duty cycle, minimum amplitude, maximum amplitude, duration, hints. Period is the time one cycle of the waveform takes (in milliseconds). Duty cycle is the ratio of the duration of the "on" time in a given period to the period given as a percentage of the period. For example, if the period is 600 milliseconds (ms) and the "on" time is 450ms then the duty cycle is 75%. Minimum amplitude is the minimum power (or range of motion) expressed as a percentage of total power (or range of motion) available. Maximum amplitude is the maximum power (or range of motion) expressed as a percentage of total power (or range of motion) available. Duration is the total time the waveform is active for, given in milliseconds. Hints are additonal directions given as text strings. Each hint is a parameter follwed by an equal sign ("=") and then the value. Each parameter=value is separated by a comma. Hints may indicate waveform shapes, hardware specific instructions or other information. Using these six parameters you can program very realistic behavior such as the actions of the bulbospongiosus and ischiocavernosus muscles during penile erection (see illustration below created from laboratory data). Both the minimum and maximum amplitude rise over time; the actual amplitude of the waveform is small (the difference between minimum and maximum amplitude). This gives a "throbbing" type of erection. To get the ascending waveform shown in the following diagram each wave would be computed individually and loaded into the queue with the **`penis.erection()`** method.



In the case of some types of movement, minimum amplitude is ignored. In **`pelvis.swivel()`** (rotational motion) and **`pelvis.wiggle()`** (side to side motion) minimum amplitude is always zero since the motion occurs on equal sides of the axis of motion (dwell point). See the section on classes/objects, Pelvis, for clarification.

The androids themselves are stationary and must be placed in position by the user. There are two reasons for this rule: (1) large body movements (force and distance) are required to move adult humans from one position to another. This increases the possibility of injury should something go wrong. (2) the computational aspect of moving a human from one position to another is a problem. It requires large amounts of raw computer power and complex programming. The extensive work on creating walking robots bears this observation out.

4. Rudimentary Nervous System

Humans are able to sense the world around them through six senses: cutaneous (touch), taste, smell, sight, hearing and balance. In Maria a limited range of senses are available: touch and hearing. The sensitivity of these sensations is determined by the hardware, software and the Maria model.

Change in a sensation is used to trigger an event (the ability to react). In the case of touch, change is a good indicator since in the normal human, the touch sensation is related to change. If we put on a hat, we're not continually reminded that we're wearing it. The only time we become aware is change in the touch sensation (i.e. the hat is blown off of our head, removal of the touch sensation). This is to be expected; trying to process all of the inputs from the nerves all the time would be impossible for our nervous system. Cutaneous sensation can be further broken down into subgroups such as:

Touch: touch, pressure, vibration
Thermal: hot, cold
Pain: discomfort, itch
Sensory blends: wetness, smoothness
(from *Mechanisms of Cutaneous Sensation* by David Sinclair, Oxford Unversity Press, 1981)

Two important concepts are part of touch or pressure sensation: detection threshold and resolution. Detection threshold is the minimum pressure that can be detected by the sensor (skin or synthetic). Resolution is the fineness with which a varying pressure can be detected or in some cases, the area over which the pressure is applied.

Detection threshold factors: rate of application (how fast the pressure rises), pressure, time of application (how long does the pressure last), temperature, skin/sensor surface (wet, rough, caloused), cutaneous innervation (number of receptors per surface area), depth (indentation of sensing surface), area of object (pointed object versus blunt object)

Resolution (fineness) factors: pressure, frequency (sampling rate) and area (sensor area)

Only the pressure sensation is available to Maria scripts, the fineness of the pressure is determined by hardware and software. Only specific areas are available for touch sensation, further localization is not available. For larger areas several sensors may be used and the results averaged. Proprioception, the sense of position of the parts of the body relative to the body may be useful in some types of sensation instead of or in conjunction with touch. For example, the position of the jaw may be easier to electronically monitor than sensors placed in the oral cavity.

The sensors used in any android built for commercial sale (as opposed to a custom-built sensor for a research lab) in the near future will provide only crude measurements when compared with the human nervous system.

With the proper hardware and software, Maria can hear and, under the proper conditions and with the right software, get the words that were spoken. It is not possible for Maria to localize the source of the speech. Maria can detect the proximity of an object; exactly how this is not specified, although in a human, vision and hearing would be used. A simple method is to use commercially available ultrasonic emitters/detectors to detect close objects.

5. Emergent Behaviors

A few very simple behaviors can combine to create emergent behaviors; that is behaviors that are not explicitly set by the programmer but emerge from the interaction of disparate behaviors through the real world (as opposed to interaction an internal model of the world). In terms of sexual interaction consider how muscular contractions of the bulbocavernosus muscle pull on the clitoris and result in more stimulation and vasocongestion. Greater vasocongestion leads to more vaginal lubrication. Depending on other variables (i.e. penis in the vagina increasing muscular tension) the total behavior can get complex very quickly but it is a result of a small number of simple, primitive behaviors. Emergent behaviors make programming simpler by eliminating the internal state model of the world. Instead of building an internal state model of the world the android reacts to the actual state of the real world as experienced through its nervous system (sensors).

6. The Maria Manifesto

A manifesto is a declaration of intentions. In the Maria Manifesto the basics of playing "nice" together between android creators and Maria script writers are outlined. Since it is impossible to predict what permutations of scripts/androids will be connected together, the manifesto outlines things in broad terms. This is not a precise document; the intent is to provide a spirit of cooperation rather than law enforcing cooperation.

Creators should not use existing instances to control new or novel features. For example, if you wish to have laser beams emiting from the eyes, don't write an interface that used the leftHand instance to control it! You'll wind up with scripts that only function with your android and cause odd behavior on other ones. Also, scripts from other users may cause odd behavior in your android. Creators should consider whether the new feature needs script control and how necessary it is in the first place. If you feel the feature is of interest and does need direct control from the script, tell us and we'll consider your request for inclusion in future updates of Maria.

Just because the script CAN control a behavior doesn't mean it has to. For example, your android may simulate breathing (chest rising and falling) but this doesn't necessarily require you to impliment the **`respirationRate()`** method to control it. What physical qualities you wish to control are up to you. However, you should make it clear to buyers/users of your android, which features exist and whether or not the script can control them. Unconcious

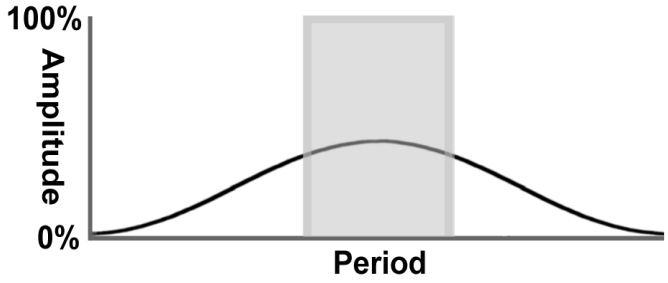
behaviors, outside of what is explicitly controlled by the script may be used to give the android a sense of humanity and avoid the uncanny valley (Mori, Masahiro "The Uncanny Valley", *Energy*, 7(4) pp. 33-35). These unconcious behaviors, while not being controlled directly by the script, may track and repond to behaviors that are controlled by the script (implicit control). For example, if your android has eyes that blink, you may chose to track the respiration rate and increase the blink rate as the respiration rate increases.

For script writers, always use the **`test`** argument to determine which items are available and controllable from the script. Although creators should provide for graceful failure at the interface/hardware level, this doesn't absolve you of responsibility to determine what physical resources the android has and to use them in an approriate manner. Methods which are not implimented in hardware will not cause any adverse reactions; the method will simply not do anything.

Script writers should also make use of the levels keyword early on in the program to determine how fine the feeling is in the android. For a simple switch the levels will be 2 with the nerve either being 0 (no touch) or 1 (touched). With sophisticated sensors you could have more than 100 levels. The behavior should be consistant whether the nerves are two levels or many levels. The fineness of the nervous system depends upon the hardware and, to some extent, the software.

Script writers should respect reasonable hardware limits. Muscles, for example, can only expand and contract below a maximum rate set by the chemistry and physics of the human body. Setting a muscle to contract at 100Hz will create unexpected behavior in the majority of androids; in most cases the software application running the script will ignore something it cannot do. Setting unrealisticly high frequencies for movement can also interfer with hardware that uses pulse-width modulation (PWM) to control the amplitude of the waveform. In general, try to avoid a period of less than 500 milliseconds (0.5 seconds).

Since biological effects are being replicated by electrical motors, the actual command given to the hardware may be different than that described in the method. This is to make up for starting inertia (motors draw more energy when starting than when constantly running) and latency (the time from the beginning of power application until something happens i.e. movement). Consider something like simple integration of the area under the curve with the voltage to the motor at maximum and an additional power factor to make up for latency. Pumps that move viscous liquids are one example of a motor that will require this treatment. See following diagram illustrating the what the script called for (a low amplitude wave with a long period) and what might be actually produced (high amplitude, short period).



If you're incorporating what most people might think of as "odd" behavior you should explicitly mention this in your description and in the keywords.

Certain minimum hardware should be expected by the script writer so that scripts can have some minimal functionality. These hardware minimums are (method, instance variable(s)):

Female Sex: **`vagina.orgasmicPlatform(), vulva.clitoris, vagina.urethralSponge`**
Male Sex: **`penis.erection(), penis.glans, penis.shaft`**
Oral Sex: **`mouth.hum(), mouth.tongue`**
Anal Sex: **`anus.sphincter(), anus.sphincter`**

It's important to create hardware which is intrinsically safe if the builder expects the user to feel comfortable and confident in using it. Here are a few simple guides to follow when designing/building hardware. ***Please note that these guidelines do not (and cannot) cover every possibility; it is up to the creator/manufacturer, the programmer and the user to ensure safe design, manufacture and use of the android or doll.***

- Don't use powerful motors or gear the motors down to create excessive torque. Keeping the power of the motors low helps protect the user from possible injury and reduces the size and cost of support components (i.e. power transistors, wiring, transformers etc.)
- Use low voltage (5 -24 volts) direct current where possible. Low voltage direct current has the least chance of causing harm in case the user comes in direct contact with it.
- Isolate all high voltage (100 volts and greater) from control lines with opto-isolators. DON'T connect it directly to the control circuit that your low voltage hardware is connected to. All high voltage lines should be grounded and properly protected from mechanical damage and wear.
- Safety should be implimented at the hardware level. By this I mean that when malfunctions occur the device should not have to ask the software to shut it down; it should do so on its own and inform the software (the software application that is running the Maria script, not the script itself) of the problem, or at least make it apparent that there is a problem (warning lights, buzzers etc.). Limit switches, current limiters for motors, fuses, thermal cutoffs etc. all can be used to keep the hardware well inside a safe operating range. This prevents a software malfunction from creating a potentially dangerous situation.
- Equipment should reset/start in an off state when power is first applied to the hardware (except where a power on condition is ALWAYS harmless). When power is shut off the hardware should not create any unsafe conditions. Consider what an electrical spike entering the control system at power up or power down might cause the hardware to do.
- Equipment should be constructed so as to be easily cleaned and maintained to prevent the build up of organic compounds which might harbor/breed bacteria and allow the transmission of infections between users of the hardware. Detailed instructions for cleaning and maintainence should be attached directly to the hardware if possible.
- All hardware must be properly grounded. Where practical the hardware must have ground fault protection.
- All sharp corners or metal edges must be removed or covered with a protective material that cannot be displaced. Any gears, belts or rotating shafts must be covered with a secure cover preventing any contact with the user, the user's clothing, jewelery or hair.
- Noxious or toxic chemicals must not be used in any hardware interacting with the user. No drugs of any type may be delivered by the machine by any method to the user. Be aware and prevent the hardware from creating an unsafe environment by the use or misuse of substances. For example, a lubricant or oil sprayed on the floor could cause serious injury if a person slipped in it.
- Hydralics and pneumatic power sources should be used with great caution. Not only are these sources more difficult to control and mechanically more powerful with the risk of greater injury during a malfunction, but pneumatic machinery could cause a life threatening air embolism should air be accidently injected into a body cavity under pressure.
- Suction used for direct stimulation (not pneumatic control) of the body should be limited to a maximum pressure of -500mm Hg (-0.7 bar). Where possible this suction should be cycled in an on/off manner to reduce the possiblity of bruising. A fluid collection device should be placed between the vacuum source and the part of the hardware physically in contact with the user to prevent body fluids from entering the vacuum source (prevent cross-contamination between users). Equipment should be designed so that the fluid collection device can be easily emptied and cleaned along with all parts of the suction/vacuum system.

How it works

The model is quite simple. The main program is a continuous event loop; the code between **`main`** and **`endmain`**.. Each time through the loop the instance variables are updated indicating the current status of the androids "nervous system". Changes in the instance variables trigger an event at the Class/Object level. The **`event`** does allow fall-through so that different combinations of Class/Objects may be used to trigger an event. Inside the event block the actual changed instance variable can be determined (if necessary). In the case of the proximity system and the speech-to-text system, the instance variables indicate only a **`true`** or **`false`**, a method has to be used to get the actual

phrase from the speech-to-text engine. To actuate parts of the android use the methods for that part or sub-part. When an actuation method is used it begins its own thread; additional actuations of the same method are queued up and run in sequence.

The structure of a Maria Script and language syntax

The script is composed of a series of statements. Whitespace (blank lines) and comments are used to make the script more readable but are discarded by the interpreter. A statement may take one or more lines but always ends with a semicolon;

Note: In the method descriptions
<level> indicates a numeric value
<void> indicates no parameters are necessary for this method.

For waveforms, six parameters are used:
`<period>,<duty>,<min>,<max>,<duration>,<hints>`
<period> period is one cycle of a waveform in milliseconds. For infinite period use a negative value.
<duty> duty cycle is the ratio of the duration of the on time in a given period, indicated as a relative value 0 to 100 (percentage), 0 = not on, 100 = maximum, on 100% of the time, 50=on 50% of the time. Default value is 50%
<min> minimum amplitude is the minimum value of the waveform relative to the maximum possible amplitude, indicated as a value 0 to 100, 0 = minimum amplitude 100 = maximum amplitude available
<max> maximum amplitude is the maximum rise of a waveform, indicated as a relative value 0 to 100, 0 = minimum amplitude 100 = maximum amplitude available
<duration> is the time over which the effect takes place in milliseconds. For continuous (until stopped) action use a negative value.
<hint> a text string of parameter=value, separated by commas (i.e. "waveform=square, vary=random"). Provides additional information about waveform shapes and hardware specific information. Hints may not be implimented in many browsers and hints will be ignored if not available. If hints are not required use **`null`** in place of the text string.

Reserved Words

```
Maria 2.01;       # Maria signature. Must be first statement in script

persist           # this keyword is used to import and export variables
                  # from the script as text when program is terminated and
                  # import the variables when the program is started again.
                  # variables declared within this block persist from
                  # one script invocation to the next.
                  # Provides a low level of persistance
endpersist        # ends export block

entityname        # quoted text of entity name
description       # holds a description of entity, quoted text only,
                  # no program code
hardware          # a description of the hardware that this program
                  # was (ideally) designed to run on
keywords          # keywords quoted text, comma separated, describing entity
                  # (include sex, sexual orientation etc.)
main              # begins main loop
endmain           # end of main loop
event             # begins event block
endevent          # ends event block
if then           # if statement, begins if block
else              # else statement, must be used inside an if block
endif             # endif, ends if block
while             # while loop statement, begins while block
```

```
endwhile          # endwhile statement, ends while block
void              # void, used in function declarations
                  # to indicate no return value
function          # a function declaration, begins function block
return            # return from a function
endfunction       # end of a function block
break             # exit from current loop one level

# Class or instance method using zero or more parameters
<Class/instance>.<method>(<parameter>, . . .);

# function call using zero or more parameters
<function name>(<parameter>, . . .);
```

Boolean Operator Types

```
NOT         # logical NOT (negation), unary operator
AND         # AND, logical operator, precedence over OR
OR          # OR, logical operator
NE          # NOT EQUAL to, equality operator
EQ          # EQUAL to, equality operator
LT          # LESS THAN, relational operator
GT          # GREATER THAN, relational operator
LE          # LESS THAN OR EQUAL to, relational operator
GE          # GREATER THAN OR EQUAL to, relational operator

# boolean operator types are capitalized for clarity. Many of the people reading
# Maria scripts are not programmers and are not familiar with the
# cryptic boolean operators used in many popular languages.
```

Arithmetic Operator Types

```
+     # addition, additive operator
-     # subtraction, additive operator
*     # multiplication, multiplicative operator
/     # division, multiplicative operator
%     # modulus, multiplicative operator
=     # assignment operator
(     # left bracket (modify precedence)
)     # right bracket (modify precedence)

# precedence from high to low: unary, multiplicative, additive, relational,
# equality, logical, assignment
# unary operators and assignment operators are right-to-left associative
# all other operators are left-to-right associative
```

Variable Types

```
number      # numeric values either integer or real (stored as real using
            # IEEE 754 Double-precision 64 bit floating-point standard)
text        # quoted text (characters)called strings in other languages
boolean     # true or false
media       # primarily implimented for storing sound files for
            # immediate use in voice applications. Accepts URL to locate
            # actual media files
```

When declaring a variable it may be given a default value. Examples

```
number tree;
number bush = 5;
text frankstate = "What is that";
```

Arrays use square brackets in the declaration to indicate size. Arrays index begins at 0.

```
number[23] dayarray;    # this is an array holding 23 numbers
dayarray[20] = 349;     # assigning a value to an array element
text[4] sarahsays;      # this is an array holding text strings
boolean[10][6] twod;    # this is a two-dimensional boolean array

# an array may be initialized with a comma separated list
# of values. Values are placed in the array in order,
# any excess values are discarded. Example below
number[5] somevals = 23.7, 89.002, 12, 45.01, 13.49;
```

Variable Scope

Variables declared outside of **`event`**, **`main`** or a **`function`** declaration are always global in scope (may be accessed/changed from anywhere in the program). Variables declared inside a function (as a parameter or in the function body) are local to that function. Variables declared inside an event are local to that event. Variables must be declared before being used.

Constants

`true`
`false`
`null`

Universal Method Arguments

These are keywords that can be used as an argument in **any** method.
`status` - is the method currently executing in a thread. The method replies with true or false
`test` - tests to see whether this method is available (hardware or software). The method replies with a true or false.
`finish` - this kills the currently executing thread (if executing) represented by the method and empties the queue. Depending upon the method replies true or false.

Script Structure

The structure of a Maria script is very simple. Following the **`Maria 2.01;`** signature line global variables can be declared/initalized, functions declared. Following this section is the **`main`** script loop. This contains all the code for the program and runs in a loop until script termination. Function declarations should not be placed in **`main`**, nor should global variables be initalized or declared in **`main`**

Events

Classes which can trigger an event are: **DateTime, Sense, Head, Mouth, Thorax, RightArm, RightHand, LeftArm, LeftHand, Breasts, Chest, Abdomen, Pelvis, MonsVeneris, Vulva, Vagina, Pubis, Scrotum, Penis, Anus, RightLeg, RightFoot, LeftLeg, LeftFoot.** Not all of these may be implimented on a particular android; the script writer, creator and end user should be aware of this and make allowances in the script for this.

Concurrency

Concurrency is handled implicitly; the script author does not need to explicitly manage it.

Class Hierarchy; Classes, Instances and Methods

Note: The class Breasts and the class Chest are the pretty much equal; only one should be used (Breasts for females, Chest for males). Pubis is the same as MonsVeneris; use class Publis for males, class MonsVeneris for females.

World (abstract class)
Instance: none
Inherits from: nothing
Inherited by: everything

DateTime
Instance: none
Inherits from: World
Inherited by: nothing
Instance Methods
`weekday(<void>)` - replies with today's name (i.e. Sunday)
`day(<void>)` - replies with today's day of month (i.e. 21)
`year(<void>)` - replies with current year (i.e. 2006)
`month(<void>)` - replies with current month (i.e. December)
`hour(<void>)` - replies with current hour, 0 to 23 (military style time, 2 digits only)
`minute(<void>)` - replies with current minute past hour-0 to 59 (2 digits only)
`runningMilliSeconds(<void>)` - replies with time in milliseconds since start of the script
`eventTimer(<number>)` - number of milliseconds to next **DateTime** event. Resets to 0 (zero, no triggering) after each event triggered. Must be reset if needed.

Variable (abstract class)
Instance: none
Inherits from: World
Inherited by: Number, Boolean, Text, Media
Instance Methods
`get(<text>)` - opens an input window on the screen displaying the `<text>`, gets an answer from stdin (keyboard) and replies with the appropriate variable. If the appropriate type of variable is not provided the input window opens again with an error message and another request.
`put(<text>)` - prints to stdout (standard output) in most cases the browser or android control program

Number
Instance: none
Inherits from: Variable
Inherited by: nothing
Instance Methods
`toText(<number>)` - replies with text from a number
`floor(<number>)` - replies with a number, rounds to the nearest integer less than or equal to the argument
`ceiling(<number>)` - replies with a number, rounds to the nearest integer greater than or equal to the argument
`sqrt(<number>)` - replies with closest root of the instance
`log10(<number>)` - replies with logarithm base 10
`logn(<number>)` - replies with natural logarithm
`power(<number>,<number>)` - replies with the number raised to the number power
`exp(<number>)` - replies with e raised to the xth power
`seed(<number>)` - seeds the random number generator, use number between 0.0 and 1.0
`random(<void>)` - replies with random number between 0.0 and 1.0
`sin(<number>)` - replies with sine
`cos(<number>)` - replies with cosine

tan(<number>**)** - replies with tangent
sinh(<number>**)** - replies with hyperbolic sine
cosh(<number>**)** - replies with hyperbolic cosine
tanh(<number>**)** - replies with hyperbolic tangent
asin(<number>**)** - replies with arc sine (inverse sine)
acos(<number>**)** - replies with arc cosine (inverse cosine)
atan(<number>**)** - replies with arc tangent (inverse tangent)

Note: All trigonometric methods use radians as the argument unit

Boolean
Instance: none
Inherits from: Variable
Inherited by: nothing
Instance Methods
toText(<boolean>**)** - outputs "true" or "false" text

Text
Instance: none
Inherits from: Variable
Inherited by: nothing
Instance Methods
toNumber(<text>**)** - replies with Number of text or NULL
length(<text>**)** - replies with length of text string with an integer, the number of characters including spaces and non-printing characters
subText(<text>,<text>**)** - replies with place if one string is found in another, 0 if sub text not found
wordCount(<text>**)** - replies with the number of words (space separated) in the text string
cutText(<text>,<number>,<number>**)** - replies with substring cut from <number> to <number> of text
match (<text>,<text> **)** - replies with a number between 0 and 100 telling how good a match was made between the text strings. A perfect match (upper and lower case) with replies with 100.

Media
Instance: none
Inherits from: Variable
Inherited by: nothing
Instance Methods
toText(<media>**)** - replies wth original URL as text string
display(<media>**)** - display media if possible to standard output

Person (abstract class)
Instance: none
Inherits from: world
Inherited by: Body Sense Intelligence BodyPart

Body
Instance: body
Inherits from: Person
Inherited by: nothing
Instance Methods
respirationRate(<rate>**)** - sets respiration rate (number) breathes per minute .
arching(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - arching of back
scent(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - sets scent (perfume) level, used to

mimic perfume vaporized by body heat and movement. Describe scent (perfume type) **`hardware`** for proper set-up
`toy(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - allows control of a toy, amplitude 0 toy is off, 100 full on. Describe and toy type in **`hardware`** for proper set-up

Intelligence
Instance: intelligence
Inherits from: Person
Inherited by: nothing
Instance Methods
`script(``<URL>`**`)`** - tries to load AI script in AIML format, replies with true if successful or false
`query(``<text>`**`)`** - query to AI engine, method replies with <text>

Note: The standard intelligence engine is based upon the ALICE AI engine

Sense
Instance: sense
Inherits from: Person
Inherited by: nothing
Instance Variables
`proximityInput` - **`true`** or **`false`** is there something near the android (within a meter or so)
`listenChanged` - **`true`** when a phrase translated, reset to **`false`** when the **`heard()`** method accessed
Instance Methods
`grammar(``<URL>`**`)`** - file describing the grammar using the *Java Speech Grammar Format* (JSGF)
`heard(``<void>`**`)`** - returns a text string of last speech heard
`voice(``<URL>`**`)`** - tries to load voice file (diphone database), replies true if successful or false
`voiceParam(``<text>`**`)`** - tries to load voice parameters, replies true if successful or false, sets speaking rate, pitch and pitch range
`volume(``<level>`**`)`** -sets sound level, 50 is normal volume, 100 is maximum volume, 0 is minimum volume
`play(``<media>`**`)`** -plays named sound clip through the entity
`say(``<text>`**`)`** - plays text through entity using text-to-speech processor (if available)
`sayDelay(``<milliseconds>`**`)`** - places a delay in the say queue to help synchonize sound and action

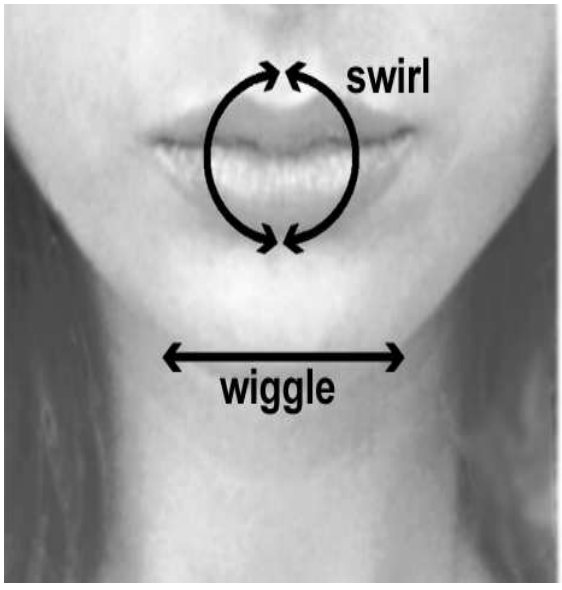
Note: The standard text-to-speech output is based on the MBROLA project, initiated by the TCTS Lab of the Faculté Polytechnique de Mons (Belgium). Input speech marked-up according to the *Java Speech API Markup Language* (JSML) . Both the text-to-speech and speech-to-text use the Java Speech API or equivalent.

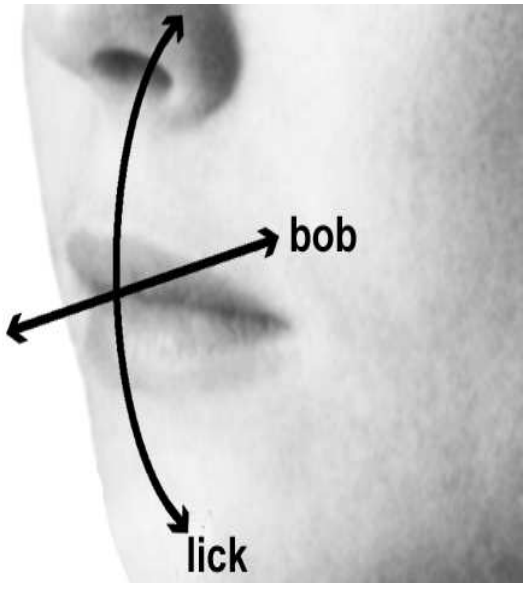
BodyPart (abstract class)
Instance: none
Inherits from: Person
Inherited by: Head, Mouth, Thorax, RightArm, RightHand, LeftArm, LeftHand, Breasts, Chest, Abdomen, Pelvis, MonsVeneris, Vulva, Vagina, Pubis, Scrotum, Penis, Anus, RightLeg, RightFoot, LeftLeg, LeftFoot
Class Variables
`self` - (the sensation for the total body part instance) sensation level at this moment
`selfChanged` - **`true`** when sensation level has changed
Instance Methods
`sensitivity(``<sensation instance variable or self>`**`)`** - replies with sensitivity (number of discrete values) available for that sensation i.e. for a simple switch activated nervous system a 2 (2 discrete values or 0 or 1). This is dependent upon the actual sensory mechanism (sensors) and is set by the program and hardware. Instance and Class variables cannot be reset or changed by the script.
`samplingFrequency(``<sensation instance variable or self>`**`)`** - replies with the actual times per second that the switch or sensor state is sampled (a measurment taken). A reply of 0 (zero) means that the system does not have a fixed rate of sampling. The sampling frequency may aid the script writer in determining if certain

conditions may or may not be detected by the androids "nervous" system (i.e. stimulation involving rapid movement such as rubbing). Each sample may or may not trigger an event; only a **change** in the value of a sensation will trigger an event. In other words, the sampling frequency is not equal to the events triggered.

Head
Instance: head
Inherits from: Bodypart
Inherited by: nothing
Instance Variables
`scalp` - sensation at this moment calculated by program
`scalpChanged` - **`true`** when sensation level has changed
`leftCheek` - sensation at this moment calculated by program
`leftCheekChanged` - **`true`** when sensation level has changed
`rightCheek` - sensation at this moment calculated by program
`rightCheekChanged` - **`true`** when sensation level has changed
`leftEar` - sensation at this moment calculated by program
`leftEarChanged` - **`true`** when sensation level has changed
`rightEar` - sensation at this moment calculated by program
`rightEarChanged` - **`true`** when sensation level has changed
`nose` - sensation at this moment calculated by program
`noseChanged` - **`true`** when sensation level has changed
Instance Methods
`eyesOpen(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - sets eyes opened or closed, max 100=open 0=closed
`lick(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - linear motion up and down; the head pivots around the atlas/occiptial bone articulation
`swirl(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - rotational movment around the axis of the mouth
`wiggle(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - side to side movement of head; the head pivots around the atlas/occiptial bone articulation
`bob(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - back and forth movement of head

Note: Use head movement in combination with mouth.





Mouth
Instance: mouth
Inherits from: BodyPart
Inherited by: nothing
Instance Variables
lips - sensation at this moment calculated by program
lipsChanged - **true** when sensation level has changed
tongue - sensation at this moment calculated by program
tongueChanged - **true** when sensation level has changed
oropharynx - the portion of the throat at the back of the mouth. Stimulation of the glossopharyngeal nerve in this area gives rise to the gag reflex in most people.
oropharynxChanged - **true** when sensation level of the oropharynx has changed
Instance Methods
fellatio(<boolean>**)** - mouth ready for fellatio. Use other methods (mouth/head) for stimulation
cunnilingus(<boolean>**)** - mouth ready for cunnilingus. Use other methods (mouth/head) for stimulation
mouthOpen(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - max open =100
tongue(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - in/out motion of tongue
nibble(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - light pressure with teeth
suck(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - slight vacuum
kiss(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - lips pursed, slight suction/pressure
hum(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - low frequency audible stimulation
salivation(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - salivation , consider a max of 50% and duty of 50% to be "standard" salivation with a continuous duration when the oral cavity is occupied

Thorax
Instance: thorax
Inherits from: Bodypart
Inherited by: nothing

RightArm
Instance: rightArm
Inherits from: Bodypart
Inherited by: nothing
massage(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - circular motion to massage chest/breasts/back
rake(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - linear (back and forth) motion. light pressure to stimulate a partner
jerk(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - hand movement at right angle to fingers, fingers closed around object

RightHand
Instance: rightHand
Inherits from: BodyPart
Inherited by: nothing
Instance Variables
thumb - sensation at this moment calculated by program
thumbChanged - **true** when sensation level has changed
indexFingerTip - sensation at this moment calculated by program
indexFingerTipChanged - **true** when sensation level has changed
middleFingerTip - sensation at this moment calculated by program
middleFingerTipChanged - **true** when sensation level has changed

Instance Methods
finger(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - motion of fingers to stimulate the genitals of partner. Hand open.
squeeze(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - squeeze fingers around object.

LeftArm
Instance: leftArm
Inherits from: Bodypart
Inherited by: nothing
massage(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - circular motion to massage chest/breasts/back of partner.
rake(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - linear (back and forth) motion. light pressure to stimulate a partner
jerk(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - hand movement at right angle to fingers, fingers closed around object

LeftHand
Instance: leftHand
Inherits from: BodyPart
Inherited by: nothing
Instance Variables
thumb - sensation at this moment calculated by program
thumbChanged - **true** when sensation level has changed
indexFingerTip - sensation at this moment calculated by program
indexFingerTipChanged - **true** when sensation level has changed
middleFingerTip - sensation at this moment calculated by program
middleFingerTipChanged - **true** when sensation level has changed
Instance Methods
finger(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - motion of fingers to stimulate the genitals of partner. Hand open.
squeeze(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - squeeze fingers around object.

Breasts
Instance: breasts
Inherits from: Bodypart
Inherited by: nothing
Instance Variables
nippleRight - includes both the right nipple and the right areola (the dark area surrounding the nipple)
nippleRightChanged - **true** when sensation level has changed
nippleLeft - includes both the left nipple and the left areola (the dark area surrounding the nipple)
nippleLeftChanged - **true** when sensation level has changed
Instance Methods
nipples(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - erection of nipples
swell(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - swelling of breasts

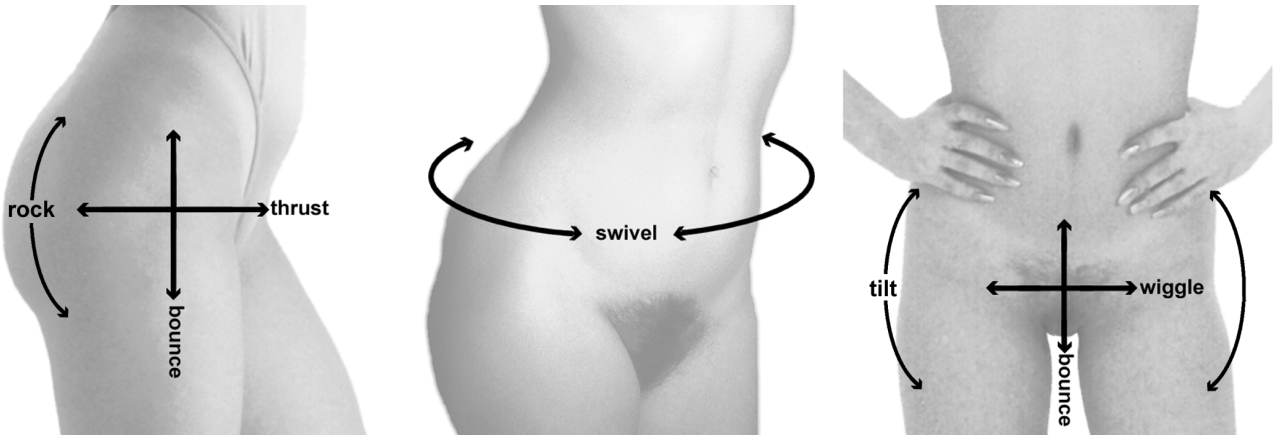
Chest
Instance:chest
Inherits from: BodyPart
Inherited by: nothing
Instance Variables
nippleRight - includes both the right nipple and the right areola (the dark area surrounding the nipple)
nippleRightChanged - **true** when sensation level has changed

nippleLeft - includes both the left nipple and the left areola (the dark area surrounding the nipple)
nippleLeftChanged - **true** when sensation level has changed
Instance Methods
nipples(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** erection of nipples

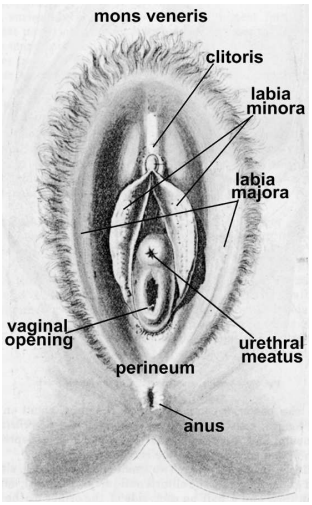
Abdomen
Instance: abdomen
Inherits from: Bodypart
Inherited by: nothing

Pelvis
Instance: pelvis
Inherits from: Bodypart
Inherited by: nothing
Instance Methods
thrust(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below
swivel(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below
wiggle(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below
bounce(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below
rock(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below
tilt(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - see description below

Note: Pelvis is used to provide motion for the sex organs which are attached to the pelvis. Bounce is a back and forth motion of the pelvis along the axis of the vagina/rectum. For the female during coitus, an amplitude of 0% bounce is the penis fully withdrawn from the vagina, 100% bounce amplitude is full insertion of the penis in the vagina. Swivel is a rotation of the hips around the axis of the vagina/rectum. Wiggle is a side -to-side motion of the hips. Thrust is a back and forth motion of the pelvis along the axis of the erect penis (standing at a right angle to the body). For the male during coitus, an amplitude of 0% thrust is the penis fully withdrawn from the vagina, an amplitude of 100% thrust is the penis fully inserted in the vagina. Rock is a motion that pivots around the lumbosacral joint front to back (along the sagittal plane). Tilt is a side to side tilting of the pelvis around the lumbosacral joint (along the coronal plane). All of these motions can be used by either sex. Since swivel,wiggle, rock and tilt movements are around a center (dwell) point, minimum amplitude is always zero (0) and any other value is disregarded. See diagram following to understand these motions.



MonsVeneris
Instance: monsVeneris
Inherits from: BodyPart
Inherited by: nothing:

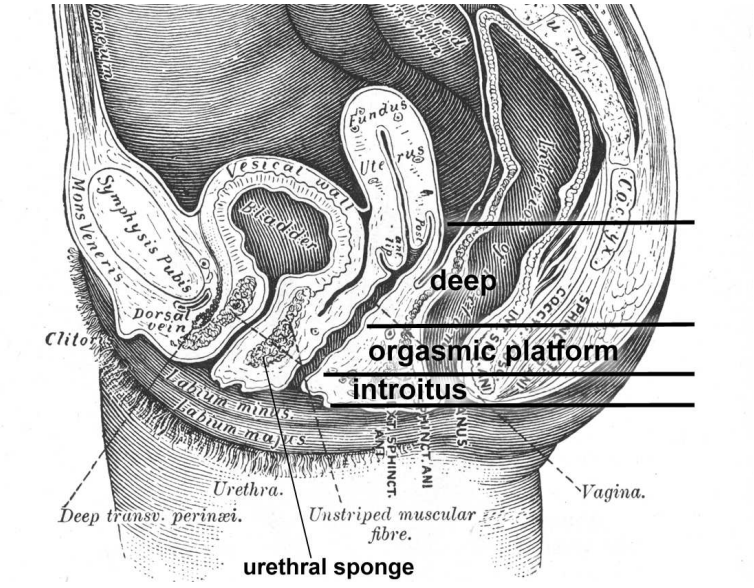


Vulva
Instance: vulva
Inherits from: BodyPart
Inherited by: nothing
Instance Variables:
labiaMajora - large external lips of the vulva
labiaMajoraChanged - **true** when sensation level has changed
labiaMinora - smaller lips of the vulva
labiaMinoraChanged - **true** when sensation level has changed
clitoris - the entire clitoris including the glans and the body
clitorisChanged - **true** when sensation level has changed
urethralMeatus - the urinary outlet, located between the glans clitoris and vagina
urethralMeatusChanged - **true** when sensation level has changed
Instance Methods:
labiaSwell(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** **-** swelling of the labia (minora and majora)
urinate(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** the discharge of urine from the urethral meatus
clitoralErection(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** erection of the clitoris

Note: For the rationale behind the method **urinate()** see the note under Vagina

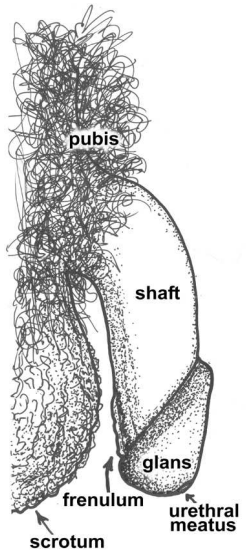
Vagina
Instance: vagina
Inherits from: BodyPart
Inherited by: nothing
Instance Variables:
introitus - the external opening of the vagina
introitusChanged - **true** when sensation level has changed
urethralSponge - the tissue surrounding the urethra, can be felt as a rounded mass in the vagina (the G-spot)
urethralSpongeChanged - **true** when sensation level has changed
deep - the area past the urethral sponge to the cervix
deepChanged - **true** when sensation level has changed
Instance Methods
tenting(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - the tenting of the vagina caused by the elevation of the uterus
vasocongestion(<period>,<duty>,<min>,<max>,<duration>,<hints>**)** - the swelling or tightness of the vagina due to increased vasocongestion of the pelvis and the smooth muscle lining the vagina
introitus(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** muscles at the opening of the vagina (ischiocavernosus and bulbocavernosus)
orgasmicPlatform(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** the muscles in the outer third of the vagina including the pubococcygeus (part of the levator ani)
deep(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** the action of the deep muscles of the vagina near the fornix
lubrication(<period>,<duty>,<min>,<max>,<duration>,<hints>**) -** lubrication of the vagina. For consistency across different models of androids a 10% amplitude (max) with a duty cycle of 50% will be standard for "normal" lubrication based on a continuous duration after sexual stimulation has been started. A max amplitude of 80% to 100% will be considered female ejaculation (min amplitude set at 0%) with individual "spurts" of 0.5 to 1.5 seconds each (adjust period and duty cycle for realism)

Note: There is no standard nomenclature for defining zones throughout the length of the vagina. I have used **`urethralSponge`** as an instance variable to indicate nerves in the "G-Spot" located on the anterior aspect of the vagina; it can be felt in the female as the spongy mass of erectile tissue surrounding the urethra (indicated on diagram). The method **`orgasmicPlatform()`** is used to indicate the first third of the vagina. The **`deep()`** portion of the vagina is the deepest two-thirds of the vagina ending at the vaginal fornix (where the cervix extends into the vagina). **`introitus()`** is the musculature at the external opening of the vagina. See the diagram to the right for clarification of these zones. Please note that these zones are only approximate and may vary from individual to individual.



There has been some controversy over so-called "female ejaculators". Some researchers claim that the expelled fluid is urine expelled from the urethra; others claim it is fluid expelled from the vagina. To provide both mechanisms the method **`urinate()`** has been provided under the Vulva class and under the Penis class (to be consistent). The method **`lubrication()`** under the Vagina class can be used like **`ejaculation()`** under the Penis class to simulate vaginal "ejaculation".

Pubis
Instance: pubis
Inherits from: BodyPart
Inherited by: nothing



Penis
Instance: penis
Inherits from: BodyPart
Inherited by: nothing
Instance Variables:
`urethralMeatus` - the opening at the tip of the penis
`urethralMeatusChange` - **`true`** when sensation level has changed
`glans` - the entire glans penis (head of penis)
`glansChanged` - **`true`** when sensation level has changed
`foreskin` - the loose fold of skin covering the glans penis
`foreskinChanged` - **`true`** when sensation level has changed
`frenulum` - the loose fold of skin underneath the glans penis
`frenulumChanged` - **`true`** when sensation level has changed
`shaft` - the shaft of the penis from the glans to the body
`shaftChanged` - **`true`** when sensation level has changed
Instance Methods:
`erection(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - erection of the penis
`urinate(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - the discharge of urine
`ejaculation(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`)`** - used for both lubrication and ejaculation, use maximum amplitude for ejaculation. For consistency across different models of androids a 10% amplitude (max) with a duty cycle of 50% will be standard for "normal" lubrication based on a continuous duration after sexual stimulation has been started. A max amplitude of 80% to 100% will be considered ejaculation (min amplitude set at 0%), with individual "spurts" of 0.5 to 2.5 seconds each (adjust period/duty cycle for realism)

Scrotum
Instance: scrotum
Inherits from: BodyPart
Inherited by: nothing
Instance Variables:
`raphe` - the "seam" running down the center of the scrotum, dividing left and right
Instance Methods:
`testesElevation(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`) -`** elevation of testes

Anus
Instance: anus
Inherits from: BodyPart
Inherited by: nothing
Instance Variables
`perineum` - area between vulva/anus or scrotum/anus (see vulva diagram)
`perineumChanged` - **`true`** when sensation level has changed
`sphincter` - the anal sphincter
`sphincterChanged` - **`true`** when sensation level has changed
`rectum` - the rectum, internal area beyond the anus
`rectumChanged` - **`true`** when sensation level has changed
Instance Methods:
`sphincter(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`) -`** anal sphincter muscle
`rectum(``<period>,<duty>,<min>,<max>,<duration>,<hints>`**`) -`** internal muscles of the rectum including the levator ani

RightLeg
Instance: rightLeg
Inherits from: Bodypart
Inherited by: nothing

RightFoot
Instance: rightFoot
Inherits from: Bodypart
Inherited by: nothing

LeftFoot
Instance: leftFoot
Inherits from: Bodypart
Inherited by: nothing

LeftLeg
Instance: leftLeg
Inherits from: Bodypart
Inherited by: nothing